CHAPTER FOUR

4.1 Introduction

This chapter presents the implementation, testing, and results of the Stock Price Prediction Web Application. It details how the logical design was mapped onto a physical platform, the development and integration of individual modules, and the testing procedures undertaken to verify, validate, and secure the system. The chapter concludes with the outcomes of the testing phase, including feedback and corresponding modifications.

4.2 Mapping Logical Design onto the Physical Platform (System Development)

The logical design developed in Chapter Three was translated into a functional system using Python and Streamlit as the core technologies. The application modules were developed using modular programming techniques to separate concerns between data handling, model prediction, and user interaction. All development was conducted in Visual Studio Code on a Windows 10 environment, with version control managed via GitHub. The entire system was then deployed to a Streamlit sharing platform for user access and feedback.

4.3 System Modules Implementation (UI, DB, etc.)

4.3.1 User Interface (UI)

The user interface was built using Streamlit, chosen for its simplicity and seamless integration with Python. It provides:

- Drop-down selections for stock symbols
- Date input widgets for specifying forecasting periods
- Real-time display of charts and predictions

4.3.2 Backend Logic

The backend includes:

- Integration with the yfinance library to retrieve historical stock data.
- Use of Facebook's Prophet model for time-series forecasting.
- Data preprocessing to ensure clean input for the model.

4.3.3 Visualization

Plotly was used to generate:

- Line graphs for stock trends
- Candlestick charts for recent activity
- Forecast plots showing future trends up to 5 years

4.4 System Modules Integration

After individual modules were successfully implemented, integration focused on ensuring seamless data flow between components:

- Historical data fetched through yfinance is preprocessed and passed to the Prophet model.
- The model output is sent to visualization functions for rendering in the UI.
- The entire flow responds dynamically to user inputs (stock symbol, date range), updating charts in real time.

4.5 Testing

Testing was conducted continuously, following Agile principles, and structured into several phases to ensure comprehensive system validation.

4.5.1 Testing Plan

The testing plan included:

- Unit Testing: Verifying individual components (data fetch, forecasting model, chart generation).
- Integration Testing: Checking how components interact.
- System Testing: Ensuring overall functionality.
- User Testing: Gaining feedback from selected end-users.

4.5.2 Verification Testing

Verification focused on ensuring the application was built according to specifications:

- Confirmed correct integration of yfinance and Prophet.
- Ensured Streamlit rendered the UI accurately.
- Verified output consistency of prediction results under known input scenarios.

4.5.3 Validation Testing

Validation ensured the app met user requirements:

- Users could input dates and select stocks.
- Charts were accurate and understandable.
- Forecasts were generated and visualized clearly.
- Feedback from finance students and novice investors validated usability and clarity.

4.5.4 System Security Testing

Basic security testing was done to ensure:

- No unauthorized access to system files
- Inputs are sanitized to prevent injection or breakage
- External data sources (Yahoo Finance API) were handled securely with error checks

4.5.5 Recommendations Made by Testers

Testers provided the following recommendations:

- Add more companies to the stock list
- Implement a data caching mechanism to speed up repeated queries
- Include a summary section explaining prediction insights
- Improve error handling when API fails

4.5.6 Responses to Recommendations from Testing

In response:

- Additional companies were added based on relevance
- Caching was partially implemented for recent queries
- A simple prediction summary was added to the UI
- Try-except blocks and user-friendly error messages were incorporated